UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Escola d'Enginyeria de Telecomunicació
i Aeroespacial de Castelldefels

MASTER THESIS

TITLE: Remote Radio Head for C-RAN

DEGREE: Master's degree in Applied Telecommunications and Engineering Management (MASTEAM)

AUTHOR: Salvador Beltrán Obiol

ADVISOR: Antoni Gelonch Bosch

DATE: 9 de juliol de 2018

Title : Remote Radio Head for C-RAN

Author: Salvador Beltrán Obiol

Advisor: Antoni Gelonch Bosch

Date: July 9, 2018

Overview

The project's main objective is to develop a Remote Radio Head (RRH) to be used under the cloud radio access networks (C-RAN) concept.

Actually, the wireless market tendency related with infrastructure implementation consist in separating the computing part of base station (BS) from the radiofrequency part. Therefore, digital signal processing is assumed in a datacenter and under the cloud computing concept. Signal samples are processed there and data streams are send to the RRH.

The project consist in developing the appropriate software layer in the RRH to interface with the virtual machine (VM) allocated in the data center.

Two different hardware has been considered to implement the RRH. In both, we followed a similar methodology, splitting the software layer in different functional modules and defining well defined implementation steps.

During project development, we addressed several test in order to verify the proper management of the involved board and check the quality and performance of the signal generated or received.

I would like to thank to all my friends and family
for helping me to attempt to focus on this project.
And specially to Mr.Blaco ,Teresa, Xavier Arteaga
and Juan Lopez for helping me when i was stucked.
And to Francisco Sanchez for the time lost helping
and advising me on the drafting of this project.
And at last but not least to my director Dr.Gelonch
who in that months of working taught me lot's of
things and tried to help with my bad working habits.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

INTRODUCTION

The objective of the project was to develop a Remote Radio Head (RRH) for the use with cloud radio access networks (C-RAN).

The concept of RRH born in the separation of the computing side from the radio equipment,basically thinking in base station(BS).

The concept lies on a centralized computing entity (cloud computing) that process all the data from different BS and sends the baseband signal to the remote radio unit.

The project consisted in developing the software interface between the radio board and the virtual machine that is executed in the Cloud-RAN.

The physical interface is based on a 1 Gb Ethernet connection and was assumed a full duplex connection to support LTE eNodeB maximum throughput.

This interface was created in C language to achieve real time transmissions and tested with 2 different setups, using different hardware (HW).

During the implementation several problems have been faced and resolved,however some of them still unresolved. Some of them where from external sources like interferences from external sources and other where implementation problems.

From the cloud, the created virtual machine generate, receive and visualize the received data.

On the project some tests where passed in order to check the quality of the generated RF signal for each setup.

CHAPTER 1. INTRODUCTION TO SOFTWARE DEFINED RADIO

In this section the software defined radio (SDR) concept will be exposed starting with a brief review of it's initials in order to understand the need of the SDR concept.

1.1. Brief SDR history

The term Software Defined Radio (SDR) can be thought as a new innovative concept on the radio technologies but the truth is that has been there for more than 30 years.

The first time that the SDR term appeared was in the E-systems company newsletter where they referred to a prototype digital base band receiver equipped with an array of processors that performed. Adaptive filtering for interference cancellation and demodulation of broadband signals, they spoke of ”Software Radio”, this was in 1984.

The concept has evolved with the DARPA's SPEAKeasy project, that began in 1991, whose primary objective was to have a single radio that could support ten different military radio protocols and operate anywhere between 2 MHz and 2 GHz and therefore to have the capability to add new protocols and modulations. At this point the SDR technologies started a exponential growing, that with some alterations, continued until now.

Joe Mitola was the first to publish on the topic of software radio, for the IEEE National Telesystems Conference in 1992, with his paper [2] “Software Radio: Survey, Critical Analysis and Future Directions”.

In 1996 the first industry association dedicated to SDR was founded as “The Modular Multifunction Information Transfer System (MMITS) Forum.” In 1998 it became the SDR Forum, and then in 2010, the Wireless Innovation Forum. The forum consisted of people and organizations from government, industry and academia. All where driven by the goal of advancing SDR-related technologies organized on committees and working groups to promote the innovation and standards on the SDR field.

In 1997 the US Department of Defense create the Joint Tactical Radio System(JTRS) with the goal of standardize and abstract layers and interfaces in order to increase interoperability. This was known as the Software Communication Architecture. It was officially canceled in 2011 by the U.S Under Secretary of Defense, who stated that the products and technologies resulting from the program were unlikely to meet the established requirements.

Nutaq (then Lyrtech) teamed up with MathWorks in 1998 for the creation of a development environment that could generate executables directly from a Simulink model for a Texas Instruments DSP and a Xilinx FPGA. This innovation intended to solve one of the biggest difficulties that was writing code for embedded processors. The DSP and FPGA were collocated on a board called the SignalMaster. It was interfaced with an A/D and D/A module and was one of the first commercial SDR development platforms.

In 2001 PSpectra evolved to GNU Radio [4] that was founded by Eric Blossom and funded by John Gilmore. GNU Radio is an open-source framework for the development of SDR applications within a PC environment. With more than 5,000 claimed users as of 2012, it is by far the most popular SDR development tool-set. Complete waveforms such as P25,

802.11, ZigBee, Bluetooth, RFID, DECT, GSM, and even LTE can be downloaded from the repository and run on any x86 system. Vanu Inc. succeeded in getting their Anywave base station approved by the FCC in 2004. The Anywave is a dual-mode base station capable of running GSM and CDMA carriers simultaneously, with all protocols layers being executed on x86 CPUs. The same year Picochip (now Mindspeed Technologies) introduced a processor designed for PHY processing (commonly referred to as a base band processor). This was PC102 that was aimed for the 3G infrastructure market. It had 308 processing elements, 14 application-specific co-processors , and could handle the MAC layer as well as other protocol elements. This kind of processors helped to reduce the size, cost, and power consumption of wireless equipment.

Texas Instruments and Xilinx joined forces on 2006 along with Nutaq (then Lyrtech) to create the first completely integrated, stand-alone SDR development platform. It was equipped with an ARM processor , a DSP, an FPGA, and a front-end tunable from 200 MHz to 1 GHz. The platform was no bigger than a shoe's box and had the capability of being battery-powered, which opened up new possibilities for out-of-the-lab applications and experiments.

Lime Microsystems unveiled its LMS6002 in 2009, a radio frequency integrated circuit (RFIC). The on a improved version named LMS6002D was at the market, being improved with the integrated data converters. The RFIC could tune anywhere between 400 MHz and 4 GHz, supported up to 28 MHz of bandwidth, and provided a selectable 16-position base band filter bank. Some years before Motorola also developed an RFIC (100 MHz to 2.5 GHz) but did not widely release it.

Between 2010-2012 for the amateur radio operators or hams realized that some of the usb digital video broadcasting - terrestrial (DVB-T) receivers that were on the market where capable of decoding the in-phase and quadrature (I+Q) components of the Rf signals.

So If they suppress the DVB-T demodulation blocks, the tuned signal was digitized and sent by the driver in RAW base band I+Q samples to the PC, where can be treated by software. This was the last SDR "boom" and the one that opened the doors of this amazing world to the hams.

Nowadays the tendency is common in most of the engineering and research areas: push for a high degree of system reconfiguration capabilities trying to obtain the maximum interoperability options lowering the cost, this can be easily seen if is compared the price of a whole new hardware with the installation of an update on the machine. What is one of the higher advantages of the SDR.

1.2. Motivation of SDR

The SDR concept has evolved from the starting adaptive filtering to a whole modifiable system, the actual definition was written for first time on the Mitola paper [2] where was exposed in the following sentence: "A software radio is a set of Digital Signal Processing (DSP). primitives, a meta-level system for combining the primitives into communications systems functions (transmitter, channel model, receiver ...) and a set of target processors on which the software radio is hosted for real-time communications."

The SDR happy idea is to reach a simple schema as shown on Fig.1.1 where is just needed

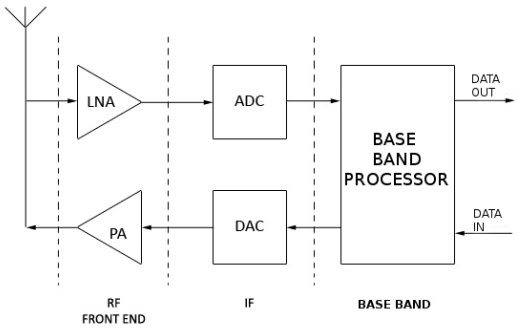

Figure 1.1: SDR concept idea. By Alba.boj [GFDL] or CC BY-SA 4.0

a simple RF front end with an ADC and a DAC with an antenna, if we check the Fig.1.2 that is the Mitola SDR schematic we can see lot's of similarities.

The actual wireless systems tend to be SDR. This is due to the high reconfiguration options that it allows. This produce a better adaptation of the systems to the new RF standards. On the other hand the new standards tend to have a higher spectral efficiency what requires from the equipment higher computational resources. This dynamic makes the hardware manufacturers to keep improving the components.

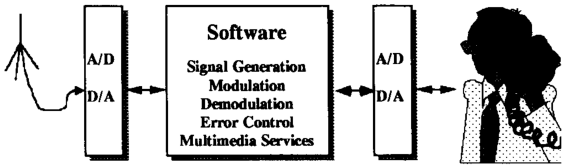

Figure 1.2: Idealized Software Radio[2]

1.3. Actual trends of SDR

As explained on the introduction the RF world is tending incorporate the SDR concept. This can be seen from 2 different points of view, the professional use and the ham use. In this section the trends of the SDR technologies in both worlds are exposed.

1.3.1. Ham trends

As was commented on the introduction to SDR section the hams get really interested on the SDR technologies round 2010, this create mainly 2 principal trends the possibility of having cheap multiband radio transceivers or increasing the capabilities of their radios.

1.3.1.1. Cheap multiband radio transceivers.

Hams started using, as explained, the dongles to receive RF communications, some of them saw there an interesting opportunity to build also a transmitter based on SDR, like the KX3 Transceiver. What allows them to get a radio with much more capabilities than standard radios and also some of them allow the software modification, what can increase the radio capabilities.

Figure 1.3: KX3 Transceiver [5]

Some of them are also Open Hardware what also permits a deeper reconfiguration or adaptation to other uses like [7]OVI40-SDR.

1.3.1.2. Increase the capabilities of their Radios.

Another possibility that requires a minimum amount of investment are the software like [6] “Ham Radio Deluxe” or [4]”GNURadio” can help the Hams to improve their radio capabilities like allowing the digital demodulation of signals.

The communication between the radio equipment and the software can be done then in different ways: The easy way is to use the radio output as the software input setting the radio speaker and the computer microphone in such configuration that allows the data transmission over the audible spectrum. This have several problems, the first one is the path noise that will introduce to the signal and also the limited BW, and the dependency of the equipment each speaker and microphone will have a different BW and sensibility at each frequency.

Another option is to communicate the radio equipment with the computer using the audible spectrum over a direct wire, this is dangerous and may burn some equipment.

As this programs has been appearing for several years the ham radios began to include some kind of standardized connection with the computer systems like USB-B or Ethernet. What is the most safe and loss less way to implement the connection.

1.3.2. Professional trends

In the professional world the SDR has been there much more time and has allowed to evolve the communications in a different way. In this subsection is described the last SDR trends on the professional field.

1.3.2.1. Command, Control, Communications, Computers, Intelligence, Surveillance and Reconnaissance (C4ISR)

SDR on the C4ISR market has a interesting paper simplifying the equipment and improving interoperability [8]. SDR was for the military systems a great opportunity to improve, due the flexibility modularity and portability that is difficult to achieve on military environments. When trying to carry multiple equipment and configure for different applications on the battlefield.

[9] With SDR multiple equipment can be integrated in a single device, what is better is the “easy” reconfiguration of the equipment and the possibility of use simultaneous signals decoders by software with the same device without loosing the security that military communications need. Also the SDR equipment is nowadays the most cost-effective solution in most of the RF fields, and the C4ISR case is not an exception.

1.3.2.2. SATellite COMmunication (SATCOM)

As it can be seen from all the sections the reconfiguration and easy integration capabilities of the SDR technologies helped lot’s of fields to improve their equipment or technologies.

In the case of SATCOM enabled the possibility of getting the transmission from most of the satellites whose encoding methods are publics, and therefore get lot of interesting information.

Starting from the meteorologic reports and passing to Satellite TV decoding to GPS signals, specially the Global Navigation Satellite System (GNSS) . CTTC has developed a

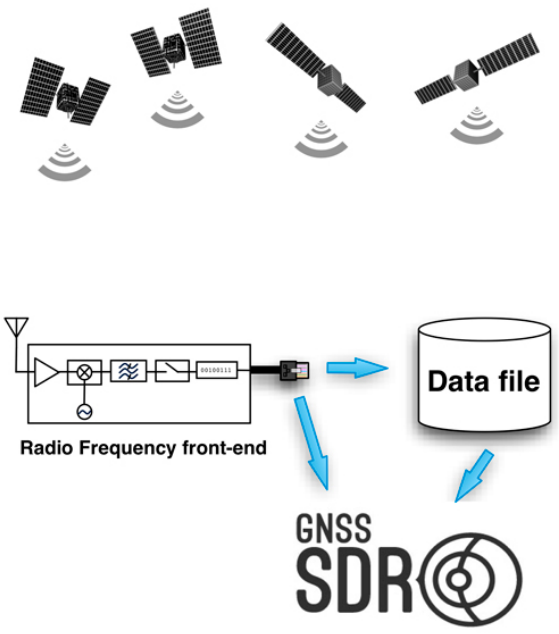

Figure 1.4: GNSS-SDR basic schema [10]

SDR open hardware receiver and the full code to receive and process the data of this set of positioning system networks.

1.3.2.3. Cognitive Radio

An interesting field of research inside the SDR possibilities is the cognitive radio (CR). CR concept is based on the idea of spectral efficiency, which objective is, as the name says, to take the major profit of the spectrum.

CR purposes an intelligent radio unit that automatically detects available channels on the wireless spectrum, and taking into account the actual communication requirements transmit the signal in one or other frequency. An interesting application for CR is the spectrum sharing , for instance, between two enterprises one of them rent the spectrum and can sub-rent his band of spectrum to another company that transmits when the first company is not using his band.

If this second enterprise sub-rent some different bands, it can ensure a minimum percentage of time to transmit the whole time. The unique inconvenient in that case is that transmitter and receiver must agree a frequency change once they detect a primary user transmission.

1.3.2.4. New Generation fronthaul Interface (NGFI)

Radio access network (RAN) should evolve to match with 5G patterns, one of the essential elements for this evolution is the Cloud RAN (C-RAN) concept, it has demonstrated [11] it's advantages on network deployment speed-up, cost saving and power efficiency.

The C-RAN concept[21] expects to decrease the costs in CAPEX and OPEX of the networks, as well as improve the radio resources management. This concept is based on the framework of Network Function Virtualization (NVF), this NVF concept consist on replacing hardware for a specific function with open software applications running on shared servers in cloud platforms.

The main idea is to have VNFs deployed on virtual machines inside a virtual infrastructure, which can be centered in a single location or distributed in several locations.

This can seem a simple question of space or price but the fact is that this C-RAN concept aims to implement the whole base-band radio signal processing in software, what means that for a later sending we are going to have some bandwidth limitations, and the most important detail, the transmission time or latency. This issues can be a hassle for real time communications.

With C-RAN the concept of base station (BS) is evolving from the two logical entities in 4G LTE, baseband unit(BBU) and remote radio unit(RRU), to a centralized BBU pool and a set of RRU that are controlled from the same BBU as seen in Fig.1.5.

[12] Some of the usual functions are shifted between RRU and BBU, what , for a better understanding leads to a renaming of the parts, BBU is renamed as Radio Cloud Center (RCC) and the RRU as Radio Remote System (RRS), the RRS coverage can be equivalent to a macro cell, and implements the Radio Aggregation Unit (RAU), that was formerly part of the BBS.

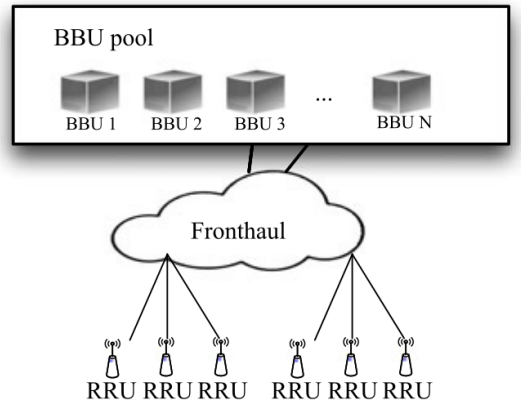

Figure 1.5: C-RAN base station scheme [11]

1.3.2.5. Radio Aggregation Unit (RAU)

The main concept relies on the RAU should be connected using a fast interface with the RRS. For this reason only I and Q samples could be transmitted to the RRS, this schema has some advantages like the easy handover control from the same point what should improve the performance in several ways as speed. This RAU will have the control of several RRS and also communication with the RCC.

[11] With this purpose a project under the encoding 1914.3 is defining the Ethernet encapsulation format for radio signal.

1.3.2.6. Remote Radio System(RRS)

The RRS is an evolution of the RRU that is also named remote radio head (RRH), and the main role of it is to transmit the signal from the BS location. In NGFI this signals are processed, as explained, from the RAU and sent as baseband IQ signals to the RRS that must process it with the right parameters and transmit the signal as also get the upstreaming communications and send them to the appropriate system as base band signals.

CHAPTER 2. PROJECT OBJECTIVE: IMPLEMENTATION OF A RRS & RAU

In this chapter the objective of the project is exposed and the used hardware is explained. In order to give to the reader a big picture of the project.

From the hardware and software limitations appears the need of taking a realistic look on the possibilities of the system. This are discussed on the next section.

2.1. RRS and RAU technical Specifications

The hardware can introduce some limitations on the system like maximum and minimum bandwidth, local oscillator frequency among others.

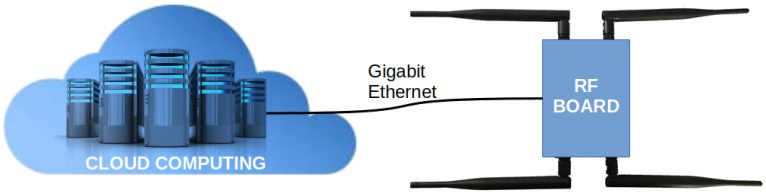

Figure 2.1: Basic interconnection schematic

The system is thought to support 4G signals on top what means that it's band width (BW must be between 1.4 to 20 MHz.

If we do the calculus of the bit rate, assuming a 16 bits per sample and taking into account the I and Q channels, and considering the different sampling frequencies we can compute the different transmission bit rates required between the datacenter and the RRH. These are shown in table 2.1.

Signal BW	Sample rate	Bit rate	Bits through Ethernet
1.4 MHz	1.92 Msps	123 Mbps	126Mbps
3 MHz	3.84 Msps	245Mbps	253Mbps
5 MHz	7.68Msps	492 Mbps	506 Mbps
10 MHz	15.36 Msps	983 Mbps	1011 Mbps
15 MHz	23.04 Msps	1475 Mbps	1518 Mbps
20 MHz	30.72Msps	1966Mbps	2024 Mbps

Table 2.1: BW vs bitrate vs Ethernet bitrate comparison

This calculations came from a prestablished number of sent samples. If a sample is encoded, as assumed with 16 bits with I and Q components then the sample size is 32 bits.

With that knowledge is easy to calculate how many samples can be sent in a single Ethernet packet. This has been calculated using a standard maximum transmission unit (MTU) of the IPV4 protocol that are 1500 bytes,[13].

Departing from there is needed to subtract from this size the IPv4 header (20 bytes), and to maintain real time we need to use UDP protocol instead of TCP due the high delay that TCP introduce on communications, what needs other 8 bytes of the header what allows a maximum allocation of 368 samples in a single packet, this will be named in the code as block size.

The bit rate that we presented on the table 2.1 is just assuming that there must be 2 different data fluxes. One for transmission and the other for reception. But when we look at the Ethernet bit rate we can see that this number is increased. This is due the headers that must be sent.

As was exposed the standard MTU for IPv4 is 1500 bytes, but an additional header is included on the transmission. This is the Ethernet header containing the physical layer addresses of sender and receiver of this Ethernet packets. This header has (14 bytes), so the second bit rate("Bits through Ethernet" column in table 2.1) is calculated taking into account this additional bits being transmitted trough the wire.

2.2. Proposed RRS Hardware

The RRS is composed from 2 components: The first one, the remote computing side, whose mission is to implement an Ethernet interface to the cloud computing. The second component is the radio one, the HW device that outputs the samples at the right frequency band.

To implement this RRS we have used 2 different setups with different HW, the first one using a ZedBoard with an ad9361 transceiver. The second setup was done using LimeSDR following the work done in the final degree project named Remote Radio Head using LimeSDR and ODROID-XU4[24].

To have a general knowledge of in which position is this hardware on the SDR market and why we have selected it, a simple comparison table was done(table2.2).

Parameter	HackRF One	Ettus B200	Ettus B210	BladeRF x40	RTL-SDR
Frequency Range	1MHz-6GHz	70MHz-6GHz	70MHz-6GHz	300MHz-3.8GHz	22MHz-2.2GHz
RF Bandwidth	20MHz	61.44MHz	61.44MHz	40MHz	3.2MHz
Sample Depth	8 bits	12 bits	12 bits	12 bits	8 bits
Sample Rate	20MSPS	61.44MSPS	61.44MSPS	40MSPS	3.2MSPS
Transmitter Channels	1	1	2	1	0
Receivers	1	1	2	1	1
Duplex	Half	Full	Full	Full	N/A
Interface	USB 2.0	USB 3.0	USB 3.0	USB 3.0	USB 2.0
Programmable Logic Gates	64 macrocell CPLD	75k	100k	40k (115k avail)	N/A
Chipset	MAX5864, MAX2837, RFFC5072	AD9364	AD9361	LMS6002M	RTL2832U
Open Source	Full	Schematic, Firmware	Schematic, Firmware	Schematic, Firmware	No
Oscillator Precision	+/-20ppm	+/-2ppm	+/-2ppm	+/-1ppm	?
Transmit Power	-10dBm+ (15dBm @ 2.4GHz)	10dBm+	10dBm+	6dBm	N/A
Price	$299	$686	$1,119	$650	$10

Table 2.2: SDR market comparison [14].

As we can see the most of boards can reach from a few MHz up to 6GHz or at least a few GHz frequency, with a wide BW at least of 3.2 MHz. In addition most of these acquisition boards includes an FPGA what facilitates its configuration and extend its usability to a wide range of applications.

Looking at what is on the market we choose the 2 explained setups.They are interesting for

different reasons. The ZedBoard setup uses an ad9361, which is a hexadecimal value of a kind of brown color but also a high performance, highly integrated RF Agile Transceiver of Analog Instruments that was specifically designed for his use on 3G or 4G base station applications. The problem of this transceiver is that it's evaluation board costs round 750 USD, which can be quite expensive if we take into account that any RF system may be composed by other several elements. In fact this is the transceiver IC that uses the Ettus B210.

For the other setup the challenge was to find another transceiver that could perform as good as the ad9361 but for a quite lower price, with that objective the LimeSDR board was chosen. It has a transceiver IC named LMS7002M this is the second-generation lime microsystems field programmable IC transceiver, and was improved to cover a wider range of frequencies and thought specifically for 2G, 3G ,4G and WiFi implementations, but a great feature of LimeSDR is that the whole board just costs 299 USD, what is less than the half of the ad9361 development board reaching same bandwidth and up to 3.6 GHz LO frequency, what makes the LimeSDR a quite competitive board.

2.2.1. ZedBoard based RRS

This setup is composed in the RRS side for a Zedboard and a FMComms3 board.

2.2.1.1. ZedBoard

This is the first element of the setup which actuates as remote processor for the remote radio, is the equivalent to the odroid on the other setup. The ZedBoard[19] is a development board based on Xilinx Zynq®-7000 and has a set of FMC (LPC) connectors that allow the communication with the FMcomms 3 board, that is the development board for the ad9361.

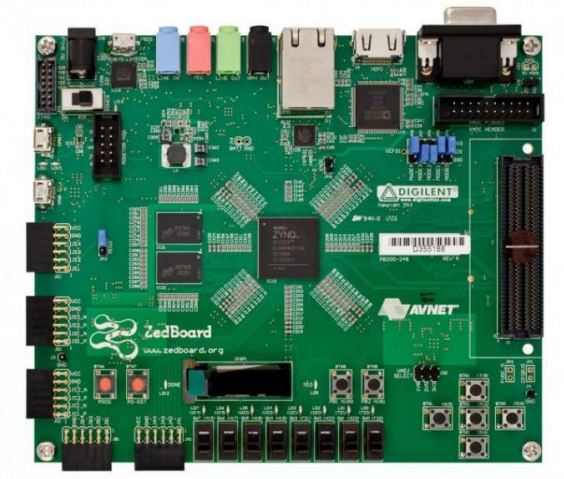

Figure 2.2: ZedBoard caption

ZedBoard detailed description

The specifications of the ZedBoard are: Zynq®7000 All Programmable SoC XC7Z020-CLG484-1
512 MB DDR3- Memory
256 Mb Quad-SPI Flash
4 GB SD card
Onboard USB-JTAG Programming
10/100/1000 Ethernet
USB OTG 2.0 and USB-UART
PS & PL I/O expansion
(FMC, Pmod™ Compatible, XADC)
Multiple displays (1080p HDMI, 8-bit VGA, 128 x 32 OLED)
I2S Audio CODEC

This board also includes an Dual ARM® Cortex™-A9 MPCore™ which is a 32 bit processor that has 2 cores with up to 667 MHz operation speed and NEON™ Processing FPU Engines powered by a 4Gb SD card with a modified version of Ubuntu by Analog Instruments.

The most interesting characteristic is, as explained the [17] FPGA Mezzanine Card (FMC) connector this is an ANSI/VITA (VMEbus International Trade Association) 57.1 standard that defines I/O mezzanine modules with connection to an FPGA or other device with re-configurable I/O capability.This kind of connector can support up to 10 Gbps communications depending on the implementation.

2.2.1.2. FMcomms3

This is a high speed module to showcase the AD9361 integrated circuit (IC), with the [18]FMcomms3 FMC port we can communicate the ad9361 to the ZedBoard. This is the RF board of the Zedboard setup. Equivalent to the LimeSDR of the other setup. Is interesting to remember that in this setup the connection between remote processing and transceiver is done with the FMC connectors that are able to reach up to 10 Gb/s.

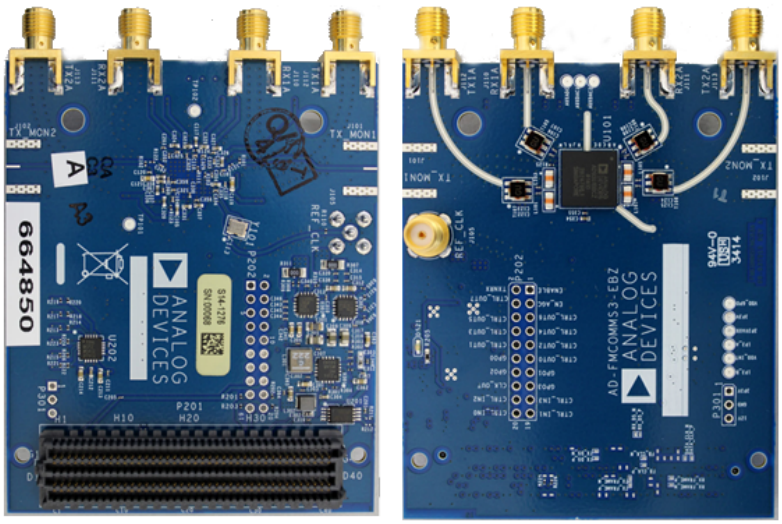

Figure 2.3: AD-FMCOMMS3-EBZ caption

FMcomms3 detailed description

RF 2 x 2 transceiver with integrated 12-bit DACs and ADCs
TX band: 47 MHz to 6.0 GHz
RX band: 70 MHz to 6.0 GHz
Supports TDD and FDD operation
Tunable channel bandwidth: ¡200 kHz to 56 MHz
Superior receiver sensitivity with a noise figure of 2 dB at 800 MHz LO
RX gain control
Independent automatic gain control
Dual transmitters: 4 differential outputs
TX EVM: ≤ -40 dB
TX noise: ≤ -157 dBm/Hz noise floor
TX monitor: ≥ 66 dB dynamic range with 1 dB accuracy
Integrated fractional-N synthesizers
2.4 Hz maximum local oscillator (LO) step size
Multichip synchronization
CMOS/LVDS digital interface

Is interesting to know that the FMComms3 board is a evolution of the FMComms2 board, so it's supposed, as it is the third iteration of the evaluation board, that the board had improved several things becaming a more stable solution. Checking the documentation can be seen that the FMComms3 board use can be quite complex, allowing to compose fir filters, interpolations, decimations and adding gain, but this configuration will be updating the driver options of the board, so it's not easily nor agile way to change it. For the configuration it must be placed inside a file and loaded to some driver setup file.

2.2.2. LimeSDR based RRS

This RRS is based in the use of a LimeSDR board for the signal acquisition and RF side plus a computer based in an ODROID-XU4 system.

2.2.2.1. LimeSDR

The LimeSDR is a open source SDR platform, that was used as the remote radio on the setup.

Lime SDR [14] have 3 different HW options, this implementation was done using the LimeSDR USB v.1.4., this board is really interesting because of it's configuration options and the high reconfigurability that allows in a simply way. Taking a look to his API [15] we can realize that there are 2 different options: Use high level or low level functions, meaning that we can just access the memory registers and the GPIO ports on the low level functions or simply use high level functions that use some buffers created by the API in order to control the memory and other functions to send the data to the TX channel.
LimeSDR detailed description

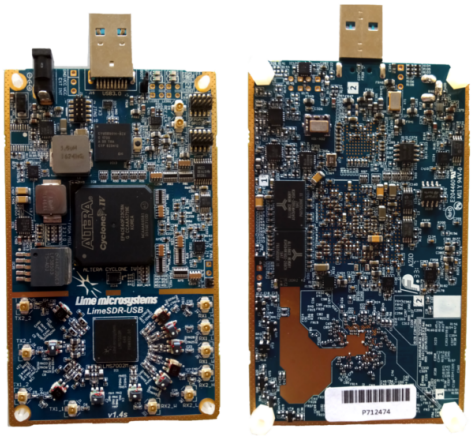

Figure 2.4: Caption of the LimeSDR v.1.4

Now the specifications of the LimeSDR USB are exposed.

RF Transceiver: Lime Microsystems LMS7002M MIMO FPRF
FPGA: Altera Cyclone IV EP4CE40F23
Memory: 256 MBytes DDR2 SDRAM
USB 3.0 controller: Cypress USB 3.0 CYUSB3014-BZXC
Oscillator: Rakon RPT7050A @30.72MHz
Continuous frequency range: 100 kHz – 3.8 GHz
Bandwidth: 61.44 MHz
RF connection: 10 U.FL connectors (6 RX, 4 TX)
Power Output (CW): up to 10 dBm
Multiplexing: 2x2 MIMO
Power: micro USB connector or optional external power supply
HDMI output connector

Is interesting to denote that there are several outputs for each reception path, filtered for a set of received frequencies, one of them matched for frequencies lower than 1.5GHz, other for frequencies higher than 1.5 GHz and the third one that can match all the frequencies supported by the Lime (100KHz to 3.8 GHz).

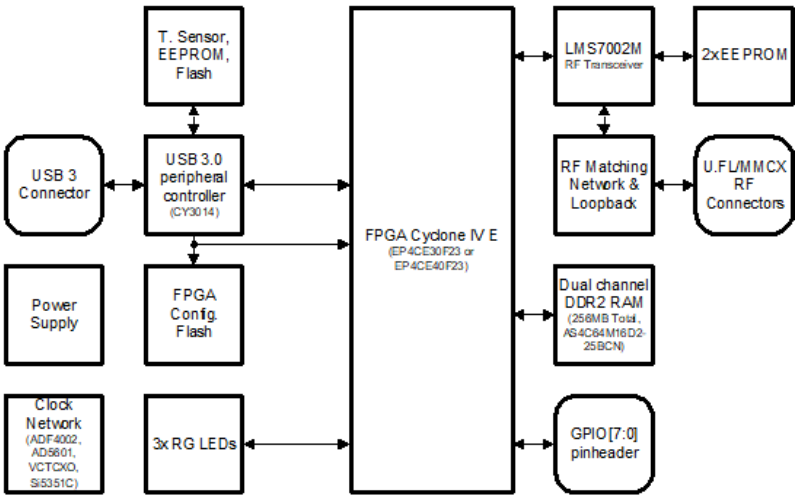

Figure 2.5: LimeSDR-USB Development Board Block Diagram [14]

2.2.2.2. Odroid

Odroid[16] is a single board computer that is the responsible on the LimeSDR setup of the remote computation. In this case the Odroid model is XU4, it executes the code that corresponds to the LimeSDR control. The Odroid implements the bridge between the remote radio and the cloud computing.

The Odroid board is connected to the LimeSDR using a USB3.1 connection which allows to send and receive the samples to/from the LimeSDR board.

This position on the setup is extremely important due is the gate to the RSS and then is the most important part to monitor as it has a clear view of the complete setup.

Knowing that the synchronization of all the system elements are driven by the radio transceiver this is the nearest controlled component and this also makes this position crucial for the system.
Odroid-XU4 detailed description

The technical specifications of the Odroid-XU4 are:

Samsung Exynos5422 Cortex^{TM}-A15 2Ghz and Cortex^{TM}-A7 Octa core CPUs
Mali-T628 MP6(OpenGL ES 3.1/2.0/1.1 and OpenCL 1.2 Full profile)
2Gbyte LPDDR3 RAM PoP stacked
eMMC5.0 HS400 Flash Storage
2 x USB 3.0 Host, 1 x USB 2.0 Host
Gigabit Ethernet port
HDMI 1.4a for display

This is quite interesting board due the need of connecting to the LimeSDR for control a great number of cores(8 cores up to 2GHz in the 32 bits processor Samsung Exynos). This could be used to parallelize threads,what is important to avoid blocking communications.

As also is interesting the USB3.0 connection due the transfer speed to the LimeSDR, what [22] is specified as 5 Gbps as maximum in the initial USB 3.0 protocol but it has increased until reach in the last revision of the standard 20 Gbps. The Odroid board uses a USB3.1

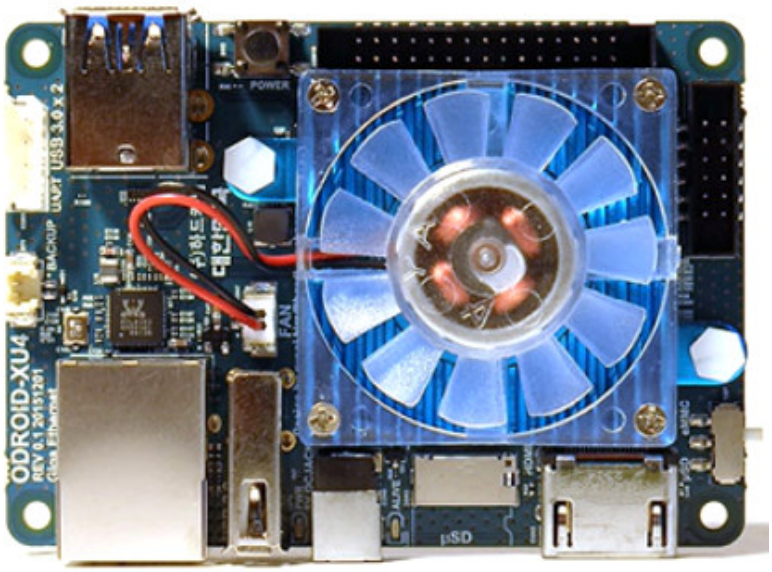

Figure 2.6: Odroid caption

standard what allows a 10Gb/s connection through the USB ports.

2.2.3. Setup comparison

To have a clear image of the comparison the table2.3 was made.
Due the use of the ad9361 instead of LMS7002M, that use lime, we can get a wider range

Board	FMComms3	LimeSDR
Frequency Range	70MHz-6 GHz	100 KHz-3.8 GHz
RF Bandwidth	61.44 MHz	61.44 MHz
Sample Depth	12 bits	12 bits
Sample Rate	61.44 MSPS	61.44 MSPS
Transmitter Channels	2	2
Receivers	2	2
Duplex	Full	Full
Interface	FMC	USB 3.1
Programmable Logic Gates	100K	40K
Chipset	AD9361	LMS7002M
Oscillator Precision	+/-2ppm	+/-1ppm Initial +/- 4ppm stable
Transmit Power	-6-0dBm	0-10dBm+
Price	$750	$299

Table 2.3: Setup RF boards comparison

of frequencies with the ZedBoard setup, reaching up to 6GHz instead of the 3.8 GHz that allows the LimeSDR. But also is interesting to see that the bandwidth can be smaller using the LMS7002M than in the ad9361, mining that the minimum BW allowed on the LimeSDR is bigger than the minimum allowed BW on the FMComms3.

After use it was found that the real bandwidth of the ad9361 can not reach the 200 KHz without interpolating samples, the minimum reached BW in the ad9361 was 2.5 MHz. Also in the LimeSDR we have done some bandwidth test due the specifications doesn't tell a minimum BW. During the tests the minimum reached BW was 2 MHz.

Another key of the system is to know it's limitations that, in terms of speed, actually are on the Ethernet connection. It's easy to see that as faster is the connection as higher the transmitted BW can be, and this is really important, because the BW of the signal can limit in several ways the possible final application that the system may be used to.

CHAPTER 3. RSS PROTOTYPE VALIDATION AND TESTS

This section presents the implementation. Giving a quick overview to the methodologies used and the processes followed to develop the basic first board codes.

3.1. Objectives of this section

This section presents the initial steps for the RRH implementation, as well as the methodology and tests used to check the proper functionality of the setup.

Also it is interesting to denote that the Zedboard setup is the most extensively explained. As already explained in previous sections, LimeSDR setup was continued from another project and therefore only a quick overview for the LimeSDR setup is exposed.

The VNF is an interface of communications between the RAU and the RSS. This is connected using gigabit Ethernet and the connection between the remote computation and the RF board (inside RSS) is as explained, two different connections, one for each setup. Due the setup requirements: on the Zedboard setup the connection is done via FMC connector and on the LimeSDR setup is done using USB3.1.

This VNF is split in 2 different codes: the cloud code (RAU) and the RSS code. It is important to have this 2 different codes even if during the test both are run on the same board because their functions are completely different. The code placed on the RAU is the responsible of generating and processing data while the code placed on the RRS is the one that is connected to the RF board. The latter is also the responsible to transmit and receive data on RF and send it to the other side as a base band signals.

3.2. Methodology

We organized project development description into several sections. They describe the development phases. The project has been divided in three main steps:

1. API test program
   In this step the idea is to get familiar with the board API and have a basic knowledge of the board capabilities.

2. Standalone VNF
   In this step the objective is to generate the Remote interface of the remote radio head and the functions to transmit the data.It needs to be tested using the Standalone version of the VNF. That requires to generate the signal samples.

3. Connect to RAU
   At this point the remote functions must be done and connected with the RSS. After that the full VNF must be implemented and therefore the final tests carried out.

The code was designed to be as modular as possible, in order to apply possible modifications, improvements or integrations as simple as possible. For that reason the design was

thought using separate independent code blocks as shown in the figure 3.1.

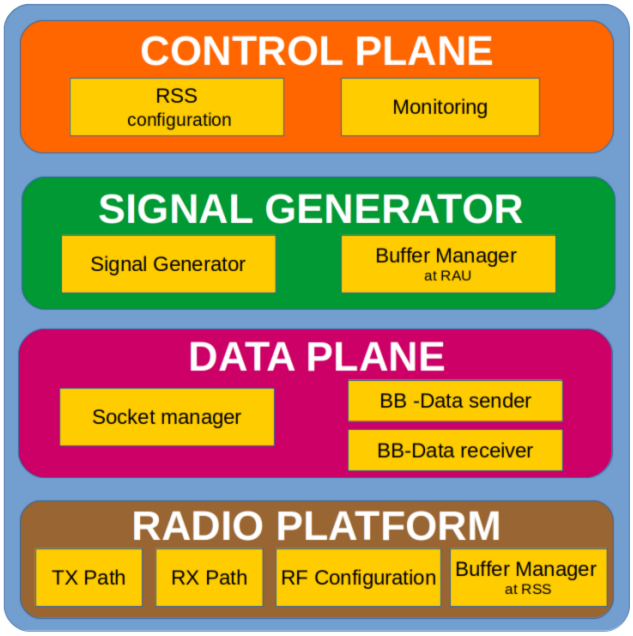

Figure 3.1: VNF general schema

As shown in figure 3.1 the general VNF schema is composed by several blocks (small rectangles painted in yellow) and also by planes (bigger rectangles painted in different colors).The blocks are somehow equivalent to functions while the planes are logic separation layers. It is important to separate the data plane from the control plane. This is because both must send useful data and motorization of the transmission at same time.

Then being separated as different layers, we can make calls to both entities from different threads if necessary. The happy idea of this schema is to have each layer as an independent thread.

This is a general schema, meaning some of the planes or functions are from the RRS side and others on the RAU side.

3.3. FMComms 3-API Initial test

To create the first test code for the Zedboard setup we used the libiio[20] library.

”Libiio has been developed and is released under the terms of the GNU Lesser General Public License, version 2. This open-source license allows anyone to use the library for proprietary or open-source, commercial or non-commercial applications. This choice was motivated by the fact that Analog Devices is a company that principally sells hardware. This library provides the clients with a better and easier way of using this hardware.”

For that first iteration, the objective is to use the board to transmit and receive RF signals. That requires to properly setup the board configuration parameters, like BW or LO frequency, to perform RF operations according our specifications.

The code implemented to fulfill this objective can be found on the appendix A. The most relevant is to know the initialization order of the different parameters, that is quite important for the correct operation of the transceiver.

We can identify several functionalities and therefore implementation stages of the code. First of all, we need to find the device, then the TX and RX paths and finally the TX and RX channels. Once the devices are created and in order to store data, we define the buffers that are linked to TX and RX.

In the moment of creating the TX and RX channels we must also configure their parameters: bandwidth, sampling frequency and the local oscillator frequency. As explained before, the board has also the capability of defining a FIR filter as well as path gain for RX or TX. Also interpolation and decimation options, that are not easy to get to, are defined. Analog Instruments released a Matlab program[23] that allows an easier configuration of the filters by configuring the parameters on the program. This generates the TX and RX configuration files.

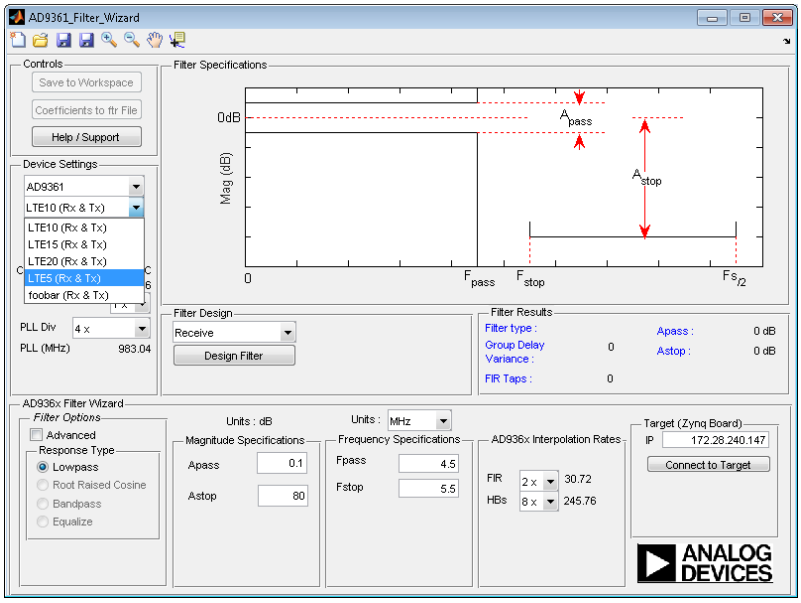

Figure 3.2: FMComms Filter Wizard caption

The code has been organized around functional blocks. Each functional block has been checked in standalone mode. First we have checked the signal generator by saving the generated samples into a binary file that we plotted using a python code (Annex B)
In order to transmit or receive, it is important to take into account the data format that the AD9361 expects. AD9361 expected binary format each I or Q word with the 12-bit signed format, but on the code the most similar format is a int 16. As the AD9361 bus-width is 12-bit, same bits as the ADC capability. Therefore Bit number 11 becomes Bit number 15. So they must be shifted by 4 and the lower 4 bits are ignored to get the expected 12 bit format.

Buffer Bit 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
AD9361 Bit 11 10 9 8 7 6 5 4 3 2 1 0 X X X X
The signal generator creates a sine table that constantly is being pushed to the TX buffer, to push the data according to the AD9361 documentation we used the following code:

```
((int16_t*)p_dat)[0] = ((int16_t)((tx_buffer[i] >> 16)&0xFFFF)); // Real (I)
((int16_t*)p_dat)[1] = (int16_t)((tx_buffer[i])&0xFFFF);         // Imag (Q)
```

As can be seen, we stored a 32 bit integer inside the table. As we need int16 samples, we apply a mask to avoid the I and Q samples to affect the other component. Applying "0xFFFF" mask we get just the first 16 bits suppressing the last 16 bits. For that reason when getting the inphase samples first we shift the 16 bits to avoid getting the last 16, and then apply the mask.

Each code piece developed has been tested by analyzing its radiofrequency behavior. The development of the code has been done connecting the board to the oscilloscope and the spectrum analyzer by a SMA to SMA cable and a T splitter. That allowed to take time and frequency measurements at same time. But once the TX code has been tested the wire has been changed for an antenna that receives the signal and sends it to the devices.

We used the signal generated by the transmitter as input for the receiver part while developing and testing the required receiver code.

We developed several tests in order to analyze the transmitter and receiver performance and the impact of specific implementations of the transmitter and receiver signal processing.

3.3.1. One tone test

The tone test basically consists in generate a tone and transmit at a specific frequency to check deviation of the signal and spurious check.

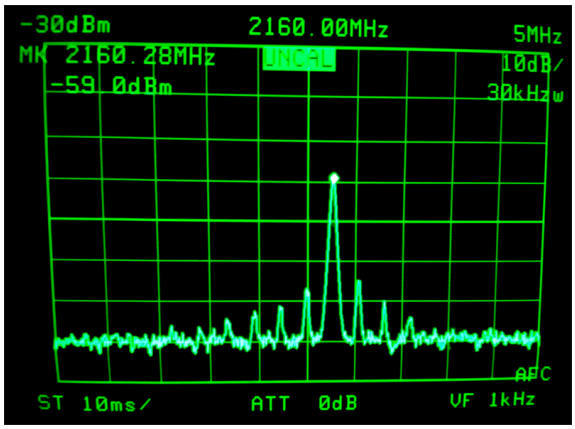

Figure 3.3: Zedboard setup with test code implementing one tone test

This is a interesting caption(fig.3.3) because it shows 2 interesting things. First is that we find out that there are harmonics, and the second is that outside the TX BW there is no signal. The figure has been taken from a sine generated at 300 KHz with a LO at 2.16 GHz and BW 3MHz. So the signal and harmonics can be seen only from 2158.5 MHz to 2161.5 MHz, as the signal frequency is 0.3 MHz higher than the LO frequency.Te lower frequency band has more harmonic frequencies than on the upper side.

We measured the power of the sine (-59 dBm) and the power of the nearest harmonic (-85

dBm), what means that the original signal is 24 dB stronger than the harmonic and round 40 dB higher than the noise.

Also was checked the periodicity of the harmonic signals that was, as expected round 300 KHz.

All the measures have been taken with a RX antenna distance of round 40 cm from the transmitter.

We also tested the FMComms3 devices using both a gain of -6 dB and 0dB. These are the two possible gains for the TX side.

3.3.2. Two tone test

The two tone test is useful to measure the third order interception point (IP3), that is an indicator for nonlinear systems and devices. The test is based on the concept of nonlinearity modeling using a low order polynomial, derived from a Taylor series expansion.

The interception point can be easily seen looking at the input vs output power of the system in a logarithmic scale. This produces a straight line with slope 1. But for the third-order nonlinear product, the output rises 3 dB at output for each dB increase at the input.

This can be measured by checking the spectrum generated with 2 tones at the input knowing that the third order inter modulation product appears at a certain combination of tone's frequencies. Being the two tones frequencies ω_a and ω_b

$$\omega_(IP3) = 2\omega_a - \omega_b \tag{3.1}$$

and because of symmetry this component also appears at $2\omega_b - \omega_a$

Therefore we can know the difference between signal power and IP3 power by measuring the difference of power between their signals. Getting $\Delta P = P_out - P_out_3$

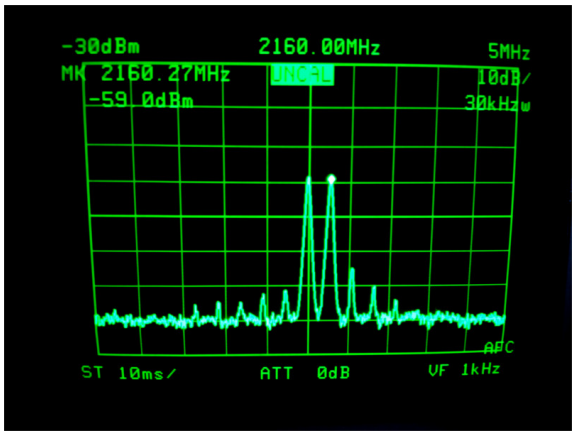

Figure 3.4: Zedboard setup with test code 2 tone test

From the caption 3.4 we can see that thee are spurious frequencies at the expected frequencies, knowing that two sine signals where transmitted at 2160.27 MHz and 2160.0 MHz.Giving a 3rd intermodulation point spurious at:

$$\omega_(IP3) = 2\Delta 2160.27 - 2160.0 = 2160.54MHz \tag{3.2}$$

With a difference of power of $\Delta P = (-59 - (-85)) = 26dBm$

Since this test is still using the ZedBoard the gain was not modified.

3.3.3. BW test

This is the last test that has been performed. The idea of this test is to check if the bandwidth set on the configuration is being transmitted. For this test a noise signal is generated and transmitted which results in an spectrum occupancy, showing a relevant power level, defined by the RF bandwidth selected by the configuration parameter.

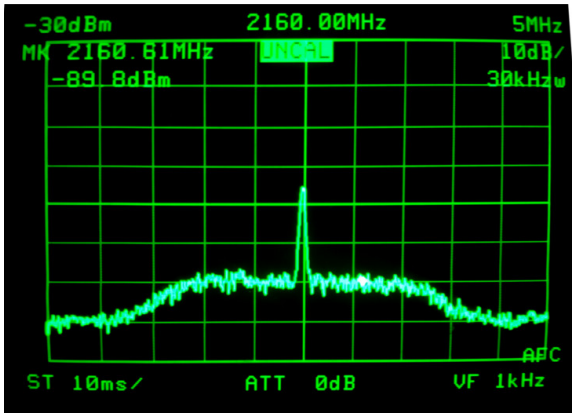

Figure 3.5: Zedboard setup with test code bandwidth test

On the caption 3.5 we can see that the BW is as expected 3MHz and centered at 2.16 GHz but some continuous signal has been introduced also on the TX. Notice that the carrier signal appears.

To know the maximum achieved BW was tested to increase the BW and reached up to 7 MHz without problems on TX ,after that frequency the signal was lost and only the carrier was appearing. Probably due the time that took the sampler to TX the signal was smaller than the time that took to the processor to generate the signal samples.

3.4. LimeSDR-API Initial test

As said before the LimeSDR setup comes from another project [24], and therefore no tests where done at this section, the test came from the other project , and simply are useful in order to compare both setups.

About the LimeSDR is interesting to know that the device must be found on first instance to get the reference and then there are 2 ways to load the configuration to the board. The first one is loading a configuration file:

```
LMS_LoadConfig(device, "/path/to/file.ini")
```

The second one is doing it manually initiating a certain TX or RX channels and setting the frequencies using the corresponding commands.

In this case, as the configuration is expected to be dynamically changing, depending on what the actual application requires. The best option is to configure the parameters manually. In order to bring the option of having a modification during the program execution avoiding a forced restart of the program.

On the case of the LimeSDR the parameters are mostly the same bandwidth, sample rate, local oscillator frequency, but in the LimeSDR the interpolation/decimation as also the gain can be easily set, just exactly as the rest of the elements.

Another important option is the auto-calibration. LimeSDR board during start up it is capable to perform an auto-calibration process, to check the proper working of the ports.

We carried out similar tests to the Zedboard ones. The results are exposed on the following pages.

3.4.1. One tone test

On the LimeSDR the tests where done by J.Cerezo and included inside [24], this tests where done with a direct connected SMA to SMA cable, with a LO frequency of 1 GHz and a BW of 2MHz.

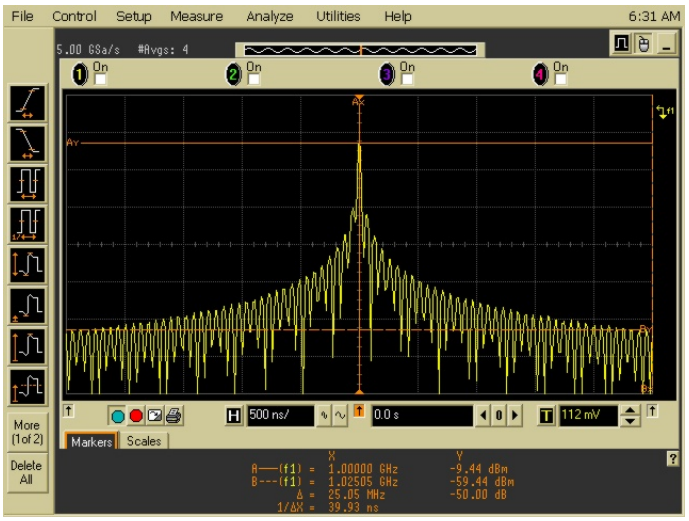

Figure 3.6: LimeSDR setup with test code one tone test

As it can be seen there is a clear sine signal, the signal has no harmonics until the gain gets to the 95% of the maximum TX gain of the LimeSDR. As the tests where done by cable connection the capture is much more clear than the ones that where done with the Zedboard setup.

Also we have to take into account that this was not taken by a spectrum analyzer it was taken with an oscilloscope with fast Fourier transform (FFT) capability, what can produce a less exact results.

3.4.2. BW test

The bandwidth test was done at the same LO frequency but using the LimeSDR maximum bandwidth to transmit.

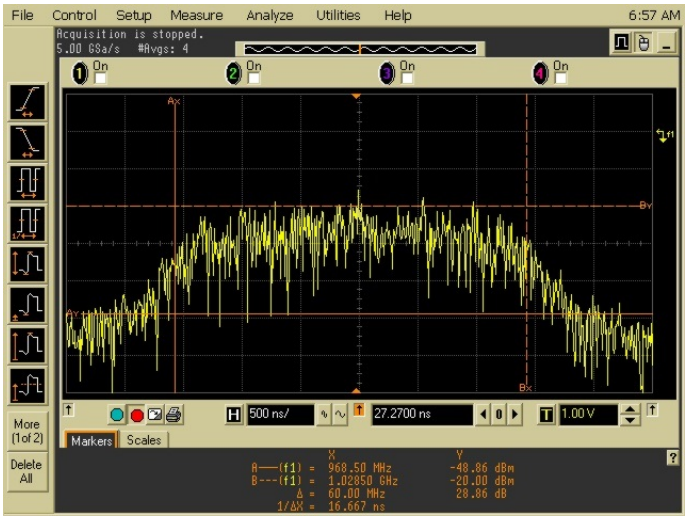

Figure 3.7: LimeSDR setup with test code bandwidth test

As can be seen from the figure it can be seen that there is a quite clear signal with a SNR of almost 30 dB.

CHAPTER 4. RSS-RAU COMMUNICATION

After a basic implementation was done on the board the idea was to define a communication schema that could be valid for almost all the boards. This was thought trying to minimize the sent parameters and taking into account that the base band samples must be processed on the cloud (RAU) and then sent to the RSS.
For this reason the figure 4.1 was followed, separating in two different paths the control

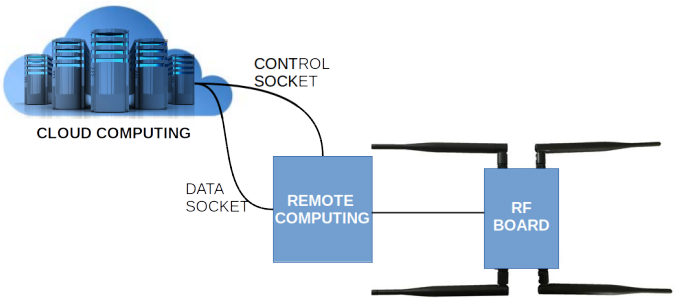

Figure 4.1: VNF connections diagram

messages and the data messages.

4.1. Objectives of this section

The 4.2 shows the states where the VNF can be. Three states where defined to keep a simple program logic.

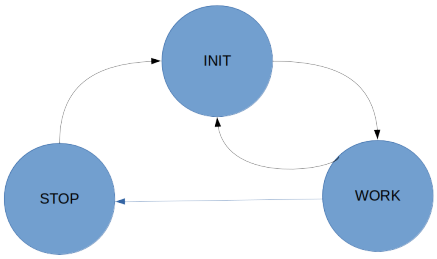

Figure 4.2: VNF stats diagram

4.1.1. Two tone test

This three states are quite simple to understand, the INIT state, is done for configuring the board, when the RSS is in INIT state is waiting for a configuration message, which contains the following parameters:

mname
STAT: State where the VNF must be on.
RRH_opMODE: Tells if must working on stand alone mode or in connected to RAU mode.

CONFIG: If any board is connected or not.
TX_CHANNELS: Number of TX channels.
RX_CHANNELS: Number of RX channels.
ENABLE_TX_CHANNEL0: Enable or disable TX channel.
ENABLE_TX_CHANNEL1: Enable or disable TX channel.
ENABLE_RX_CHANNEL0: Enable or disable RX channel.
ENABLE_RX_CHANNEL1: Enable or disable RX channel.
TX_CARRIER_FREQ.
RX_CARRIER_FREQ.
block_length: Number of samples sent on the packets.
SamplingRate: Sampling rate in Hz to set BW.
OverSample: oversampling values.
TX0_GAIN: 0.0 to 1.0 values.
TX1_GAIN: 0.0 to 1.0 values.
RX0_GAIN: 0.0 to 1.0 values.
RX1_GAIN: 0.0 to 1.0 values.
debugg: enables some helps for debugging left on the code.

As can be seen the parameters are basically the corresponding frequencies and the channels that must be activated, as also the gain and interpolating value. Another interesting parameter that is transmitted is the stat where the RRS must be set on, and the operation mode, that can basically be standalone or connected to RAU. And also if there is some board connected or not having a functional VNF or just checking the connectivity between the 2 sides of the VNF, this option was implemented as the ”CONFIG” parameter.

On the running stat the RSS is receiving the base band signal to transmit from the RAU, which compute the signal and sends the samples. The RSS sends the signal doing the convenient arrangements(modulate at LO freq, filtering, interpolation,..) to meet the configured parameters.

On the running stat the RRS transmit to the RAU some control parameters it does the same sometimes as a response from the messages that the RAU sends to the RRS, the parameters that the RRS. Transmit are :

mname
STAT: State where the RRH is.
RRH_opMODE: Tells if is working on stand alone mode or in connected to RAU mode.
CONFIG: If is set to work with RF board or not.
TXbufferROOM: Indicate the remaining space in TX buffer in samples.
ReadFlag: 0: Read, 1: Recently updated. To be read.
Flow control.
SendDataCH0: Indicates if needs or not data more data to TX0.
SendDataCH1: Indicates if needs or not data more data to TX1.

The ”STAT” and ”2RRH_opMODE” as also the ”CONFIG” parameters are designed as a response for the RAU messages and are sent to check if the RRS is in the right mode and if took the proper configuration.

Once the whole VNF was implemented a attempt of flow control mechanism has been incorporated. It is based in the use of parameters A and B to allow RRS to ask for new

data when transmission buffer is empty.

From the RUN stat it can also go to the STOP stat, this STOP status stops the RF board of the RSS by destroying all the structures(buffers, paths, devices,...) that have been created.

4.2. Methodology

As in the previous sections the program has been done by separating the code into functions, and implementing the functions separately and joined together once several pieces have been checked. The most remarkable part on this section is to see that in the VNF just one file has changed from one setup to other, this corresponds to the file that contains the specific functions for the board control. As is obvious, as there is not the same board on both setups, the code to control it must change, but following the same philosophy. That means is not necessary to redo the whole code It needs to change the code related with the use of the API of the new board. This allows a easy addition of new supported hardware.

4.3. Implementation

The implementation has separated the VNF code as explained in several files, the most important are briefly explained in this section, but the whole list can bee seen in figure 4.3. Is interesting to know that the header files have not been taken into account and the files that are outside the layers are there because have not a specific or a more centered area (referring to the files shown on fig 4.3).

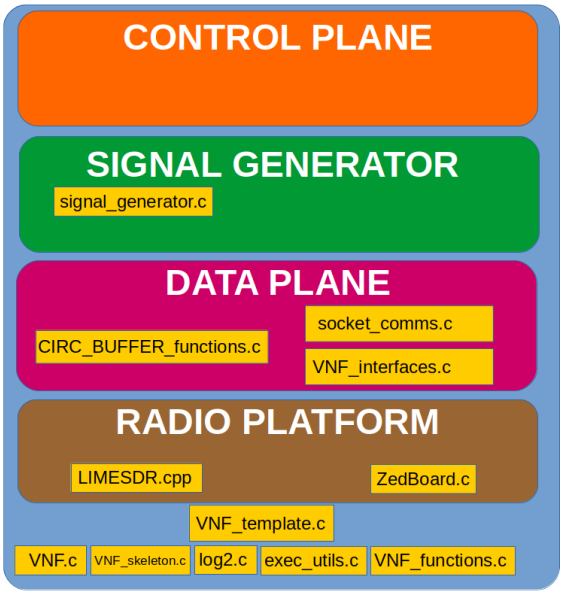

Figure 4.3: Main code classified by his layer category

For the code dealing with the board setup has been modified to work inside the whole VNF structure of files and then tested using the standalone mode, that does not depend on the

RAU. So it was implemented just the RRS side of the VNF, on both setups. ZedBoard setup had some problem that was not handled, and started having troubles with the transmission. Configuration was checked,even being correctly configured the stup never transmitted a real signal. It just transmit the carrier signal as can be appreciated on the figure 4.4.

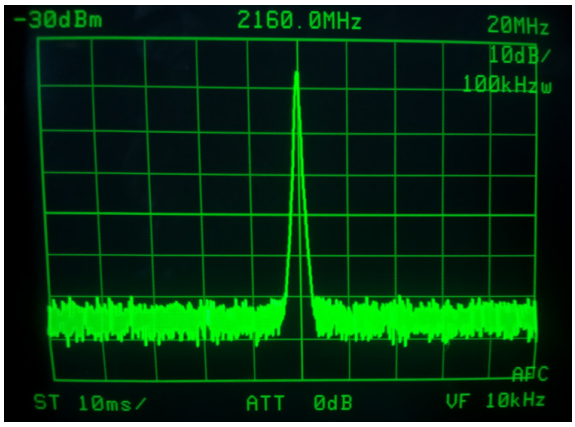

Figure 4.4: ZedBoard setup transmission attempt with the RRS Standalone code

In the figure 4.4 the ZedBoard setup was configured with a LO freq of 2.16 GHz and a BW of 3 MHz the idea was to transmit a tone of 300 KHz, but the board never transmitted anything else than the carrier, several attempts where done to find the error but no clear conclusion was extracted. Nevertheless probably it's a problem of real time data transmission. If the board process the samples faster than they are generated, since we are on standalone, the board shows all the time the carrier due the transmitted signal has been transmitted but just for a moment and then it just transmit zero's.

On the standalone step the LimeSDR worked properly so since this point the ZedBoard was left behind.

Then the RAU side of the VNF has been implemented and tested.

The complete setup with the LimeSDR with the complete functional VNF was tested as can be seen on the following subsections.

4.3.1. Tone test

Here the test consisting on transmitting a single tone was repeated again with the same parameters as in the test done in the ZedBoard with the test code. This parameters where a sine generated at 200 KHz with a LO at 2.16 GHz and BW 2 MHz.

As can be seen there are harmonics on the figure 4.5 appearing at the multiples of the sine freq, what can be expected, but a deeper look also can show the existence of a spurious between the original tone and the first harmonic. This caption was forced by setting to the maximum value the LimeSDR TX gain. For lower gain values a clean tone can be seen.

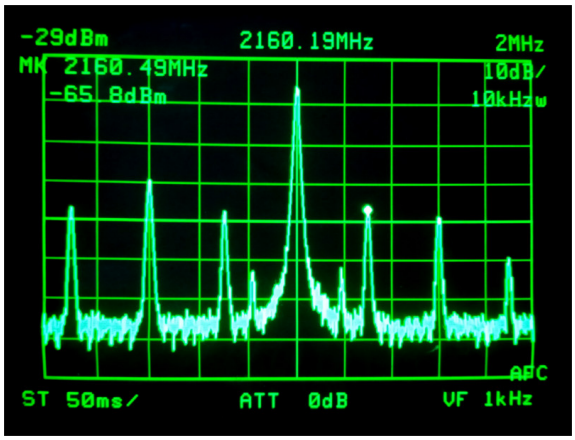

Figure 4.5: LimeSDR setup one tone test with the complete VNF

4.3.2. Two tone test

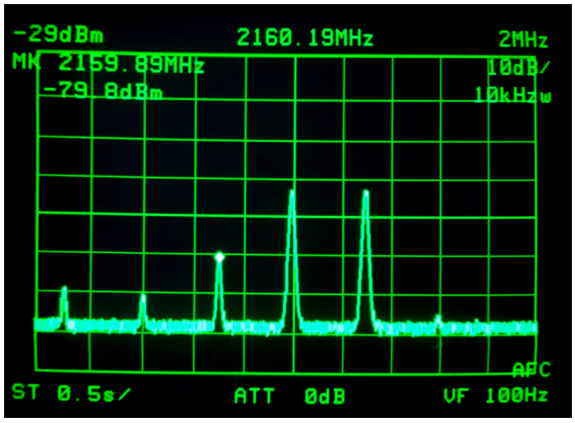

Figure 4.6: LimeSDR setup two tone test with the complete VNF

From the caption 4.6 we can see that thee are spurious frequencies at the expected frequencies, knowing that two sine signals where transmitted at 2160.2 MHz and 2160.49 MHz.Giving a 3rd inter-modulation point spurious at:

$$\omega_{(}IP3) = 2\Delta 2160.19 - 2160.49 = 2160.89MHz \tag{4.1}$$

With a difference of power of $\Delta P = (-62 - (-79)) = 17dBm$

With 2 tones at the 50% of output gain the IP3 have a power of -87 dBm and is the unique significant intermodulation product, the rest are below noise level. Decreased down to 20% where the SNR was -92 dB and the IP3 still there with a power of -88 dBm.

4.3.3. Problems

During the implementation several problems have appeared most of them have been solved but some of them still present on the setups, in this section the most important problems will be discussed and some solutions are purposed.

When the VNF was being implemented the boards had some problems with the data storage. So for the setups different solutions where adopted, for the ZedBoard an internal cyclic buffer was used, and on the LimeSDR some FIFO buffers where used.

Also at some point the defined frequency of the tests was at 2.0 GHz and a interfering signal appeared, for this reason the rest of the test where done inside a home made Faraday box, that isolated quite good and allowed to follow the rest of the tests.

Some data packets where lost during the transmissions. It was not really clear until a packet control mechanism was implemented.

This packet control mechanism was based on setting a number of sequence for each packet on TX and RX sides and when a new packet has arrived checked if the sequence number was the expected if not it just log the lost to avoid a lost of real time. This was solved by increasing the number of samples sent per packet. This provoked a fragmentation on the IPv4 packets but the Ethernet driver joint at the reception the packet again becoming a transparent fragmentation for the VNF. The problem is that now, when a packet is lost, much more data is lost. But in general terms the packet losses are minimum less than a loss per minute, and the final samples per packet was increased from the 367 calculated to 7300. The 7300 value was getting aggregating 10 packets until the packets stop loosing and we got the minimum number of aggregation to avoid losing the real time capability.

And the most important problem that was found was a synchronization problem. In the full setup three different boards are operating with the same data, but they don't share the same clock source the samples where sent and received at different frequencies and then the buffer couldn't get the samples at the right moment.

To solve that a sleep function was used, to be more precise a nanosleep, nanosleep function stops the execution of the process for the nanoseconds that we give as a parameter to the function. Nanosleep was used and some correction factors where applied to the RAU ans RSS computation board to get the same rate of sample transmission and reception.

To set the nanosleep some values where tested until get a similar periods of sending and receiving packets on the both sides of the VNF.This times where based on a initial calculus of the time that a packed needs to be processed.

A posterior attempt was to create a flux mechanism that gives the RRS the ability to ask for samples when the buffer started to empty but it didn't worked, because at the moment of sending samples the RAU transmitted them quite fast that the RRS had no time to process the anterior received samples and start loosing packets.

CONCLUSIONS

The project objective was to implement a complete RRH meaning both sides the RRS and the RAU, this was implemented nevertheless the real time was not fully achieved and still some problems to solve.

This is a good base to keep working on the VNF and could be improved and tested to became a 5G testbed.

The most of the interfaces where designed as simple entities and the pieces of code as separated as it could be, that can allow a easy reconfiguration of the VNF schema if needed in a future.

The synchronization problem have been a pain in the ass since started working with the complete VNF setup. This needs to take another point of view like implementing on the network a independent board with a stable source clock and use it with the precision time protocol (PTP), or a good flux control for the data packets.

This project faced several difficulties like the lack of documentation of both boards, as the SDR systems have small details the boards can have too much parameters and not all of them are equally documented, or found interesting.

For the ZedBoard the documentation was not bad, looked quite interesting and there is a huge community using similar setups on Analog engineer zone. But as the API (libiio) is a open source project what makes that sometimes some functions are not fitted enough or are designed in a not clear way to be used.

BIBLIOGRAPHY

[1] INNOVATIONS: Nutaq. A short history of software-defined radio (SDR) technology. [online] Available at: https://www.nutaq.com/blog/short-history-software-defined-radio-sdrtechnology.[Accessed 2 May 2018].

[2] Mitola, J. : "Software radios-survey, critical evaluation and future directions". *Telesystems Conference*.(NTC-92.),(1992) ix, 3, 4, 5

[3] Primidi. Software defined radio history. [online] Available at: https://www.primidi.com/software-defined-radio/history. [Accessed 5 May 2018].

[4] Radio, G. N. U. "The gnu software radio." [online] Available at: https://gnuradio. org [Accessed 5 May 2018]. 3, 6

[5] Elecraft Inc:KX3 Transceiver specifications.[online] Available at: http://www.elecraft.com/KX3/kx3.htm[Accessed 5 May 2018]. ix, 6

[6] HRD Software, LLC: Ham radio deluxe.[online] Available at: http://ham-radio-deluxe.com/ [Accessed 7 May 2018]. 6

[7] Elektronikreparaturen aller:SDR-Transceiver "OVI40" [online] Available at: https://www.electronicrepair.de/ovi40-order-page [Accessed 7 May 2018]. 6

[8] Stephanie Chiao:The impact of software-defined radio on C4ISR. [online] Available at: https://www.c4isrnet.com/opinion/the-compass/2016/06/03/the-impact-of-software-defined-radio-on-c4isr/ [Accessed 15 May 2018]. 7

[9] Michael Peck:'Game changer': Army wants an unmanned electronic warfare vehicle. [online] Available at: https://www.c4isrnet.com/unmanned/robotics/2017/05/23/game-changer-army-wants-an-unmanned-electronic-warfare-vehicle/ [Accessed 15 May 2018]. 7

[10] GNSS-SDR, CTTC,[online] Available at: https://gnss-sdr.org/ [Accessed 15 May 2018]. ix, 7

[11] Chih-Lin, I., Li, H., Korhonen, J., Huang, J., and Han, L. "RAN Revolution with NGFI (xHaul) for 5G". *Journal of Lightwave Technology*.**36**(2), 541–550. (2018) ix, 8, 9

[12] Charlie Ashton, Wind River, Ltd.: "Will New CRAN Architecture be the Trigger for Widespread Deployments?" [online] Available at: http://blogs.windriver.com/wind_river_blog/2016/02/will-new-cran-architecture-be-the-trigger-for-widespread-deployments.html [Accessed 20 May 2018]. 8

[13] Information Sciences Institute and University of Southern California : Internet Protocol, Darpa internet program. [online] Available at: tools.ietf.org/html/rfc791#page-11 [Accessed 21 May 2018]. 11

[14] Lime Microsystems Ltd: Lime SDR official webpage. [online] Available at: https://myriadrf.org/projects/limesdr/ [Accessed 10 Jun 2018]. ix, xi, 12, 15, 17

[15] Lime Microsystems Ltd: Lime SDR official API documentation webpage. [online] Available at: http://docs.myriadrf.org/LMS_API/index.html [Accessed 10 Jun 2018]. 15

[16] ODROID, INC : Odroid main webpage.[online] Available at: odroidinc.com/collections/odroid-single-board-computers/products/odroid-xu4 [Accessed 15 Jun 2018]. 17

[17] Devlin, Malachy, VITA Technologies : "VITA 57 (FMC) opens the I/O pipe to FPGAs". [online] Available at: http://vita.mil-embedded.com/articles/vita-fmc-opens-io-pipe-fpgas/ [Accessed 17 Jun 2018]. 14

[18] Analog Devices, Inc: AD-FMCOMMS3-EBZ product main page [online] Available at: http://www.analog.com/en/design-center/evaluation-hardware-and-software/evaluation-boards-kits/EVAL-AD-FMCOMMS3-EBZ.html#eb-overview [Accessed 20 Jun 2018]. 14

[19] Avnet, Inc: ZedBoard product main page [online] Available at: zedboard.org/product/zedboard [Accessed 30 Jun 2018]. 13

[20] Avnet, Inc: libiio documentation main page [online] Available at: http://analogdevicesinc.github.io/libiio/ [Accessed 20 Jun 2018]. 22

[21] Rodriguez, Veronica Quintuna and Guillemin, Fabrice: "Cloud-RAN modeling based on parallel processing" *IEEE Journal on Selected Areas in Communications*. **36**(3), 457–468. (2018) 8

[22] Saunders, Brad; Nardozza, Liz (25 July 2017). "USB 3.0 Promoter Group Announces USB 3.2 Update". USB.org. USB 3.0 Promoter Group. Retrieved 27 July 2017. 17

[23] Analog Devices Inc. FMComms filter Wizard for Matlab main page. [online] Available at: https://wiki.analog.com/resources/eval/user-guides/ad-fmcomms2-ebz/software/filters [Accessed 10 Jun 2018]. 23

[24] Jordi Cerezo: Remote Radio Head using LimeSDR and ODROID-XU4. 12, 26, 27

APPENDICES

APPENDIX A. ZEDBOARD BOARD TEST CODE USING LIBIIO

```
1 /*
2  * AD9361- FMCOMMS 3 with  ZedBoard  testing example
3  *
4  * This library is free software; you can redistribute it and/or
5  * modify it under the terms of the GNU Lesser General Public
6  * License as published by the Free Software Foundation; either
7  * version 2.1 of the License, or (at your option) any later version.
8  *
9  * This library is distributed in the hope that it will be useful,
10 * but WITHOUT ANY WARRANTY; without even the implied warranty of
11 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
12 * Lesser General Public License for more details.
13 *
14 **/
15 #include <stdlib.h>
16 #include <stdbool.h>
17 #include <stdint.h>
18 #include <string.h>
19 #include <signal.h>
20 #include <stdio.h>
21 #include <math.h>
22 #include <complex.h>
23 #ifdef __APPLE__
24 #include <iio/iio.h>
25 #else
26 #include <iio.h>
27 #endif
28 //#include </home/analog/Desktop/libiio/iio-private.h>
29
30 /* helper macros */
31 #define KHZ(x) ((long long)(x*1000.0 + .5))
32 #define MHZ(x) ((long long)(x*1000000.0 + .5))
33 #define GHZ(x) ((long long)(x*1000000000.0 + .5))
34 #define M_PI 3.14159265358979323846
35 #define ASSERT(expr) { \
36     if (!(expr)) { \
37         (void) fprintf(stderr, "assertion failed (%s:%d)\n", __FILE__,
              __LINE__); \
38         (void) abort(); \
39     } \
40 }
41
42 /* RX is input, TX is output */
43 enum iodev { RX, TX };
44
45 /* common RX and TX streaming params */
46 struct stream_cfg {
47     long long bw_hz; // Analog banwidth in Hz
48     long long fs_hz; // Baseband sample rate in Hz
49     long long lo_hz; // Local oscillator frequency in Hz
50     const char* rfport; // Port name
51 };
52
```

```
53 /* static scratch mem for strings */
54 static char tmpstr[64];
55
56 /* IIO structs required for streaming */
57 static struct iio_context *ctx   = NULL;
58 static struct iio_channel *rx0_i = NULL;
59 static struct iio_channel *rx0_q = NULL;
60 static struct iio_channel *tx0_i = NULL;
61 static struct iio_channel *tx0_q = NULL;
62 static struct iio_buffer  *rxbuf = NULL;
63 static struct iio_buffer  *txbuf = NULL;
64
65 static bool stop;
66
67 /* cleanup and exit */
68 static void shutdown()
69 {
70     printf("* Destroying buffers\n");
71     if (rxbuf) { iio_buffer_destroy(rxbuf); }
72     if (txbuf) { iio_buffer_destroy(txbuf); }
73
74     printf("* Disabling streaming channels\n");
75     if (rx0_i) { iio_channel_disable(rx0_i); }
76     if (rx0_q) { iio_channel_disable(rx0_q); }
77     if (tx0_i) { iio_channel_disable(tx0_i); }
78     if (tx0_q) { iio_channel_disable(tx0_q); }
79
80     printf("* Destroying context\n");
81     if (ctx) { iio_context_destroy(ctx); }
82     exit(0);
83 }
84
85 static void handle_sig(int sig)
86 {
87     printf("Waiting for process to finish...\n");
88     stop = true;
89 }
90
91 /* check return value of attr_write function */
92 static void errchk(int v, const char* what) {
93      if (v < 0) { fprintf(stderr, "Error %d writing to channel \"%s\"\nvalue
           may not be supported.\n", v, what); shutdown(); }
94 }
95
96 /* write attribute: long long int */
97 static void wr_ch_lli(struct iio_channel *chn, const char* what, long long val)
98 {
99     errchk(iio_channel_attr_write_longlong(chn, what, val), what);
100 }
101
102 /* read attribute: long long int */
103 static void rd_ch_lli(struct iio_channel *chn, const char* what, long long
       *val)
104 {
105     errchk(iio_channel_attr_read_longlong(chn, what, val), what);
106
107 }
108
```

```
109 /* write attribute: string */
110 static void wr_ch_str(struct iio_channel *chn, const char* what, const char*
        str)
111 {
112     errchk(iio_channel_attr_write(chn, what, str), what);
113 }
114
115 /* helper function generating channel names */
116 static char* get_ch_name(const char* type, int id)
117 {
118     snprintf(tmpstr, sizeof(tmpstr), "%s%d", type, id);
119     return tmpstr;
120 }
121
122 /* returns ad9361 phy device */
123 static struct iio_device* get_ad9361_phy(struct iio_context *ctx)
124 {
125     struct iio_device *dev =  iio_context_find_device(ctx, "ad9361-phy");
126     ASSERT(dev && "No ad9361-phy found");
127     return dev;
128 }
129
130 /* finds AD9361 streaming IIO devices */
131 static bool get_ad9361_stream_dev(struct iio_context *ctx, enum iodev d,
        struct iio_device **dev)
132 {
133     switch (d) {
134     case TX: *dev = iio_context_find_device(ctx, "cf-ad9361-dds-core-lpc");
            return *dev != NULL;
135     case RX: *dev = iio_context_find_device(ctx, "cf-ad9361-lpc");  return
            *dev != NULL;
136     default: ASSERT(0); return false;
137     }
138 }
139
140 /* finds AD9361 streaming IIO channels */
141 static bool get_ad9361_stream_ch(struct iio_context *ctx, enum iodev d, struct
        iio_device *dev, int chid, struct iio_channel **chn)
142 {
143     *chn = iio_device_find_channel(dev, get_ch_name("voltage", chid), d == TX);
144     if (!*chn)
145         *chn = iio_device_find_channel(dev, get_ch_name("altvoltage", chid), d
                == TX);
146     return *chn != NULL;
147 }
148
149 /* finds AD9361 phy IIO configuration channel with id chid */
150 static bool get_phy_chan(struct iio_context *ctx, enum iodev d, int chid,
        struct iio_channel **chn)
151 {
152     switch (d) {
153     case RX: *chn = iio_device_find_channel(get_ad9361_phy(ctx),
            get_ch_name("voltage", chid), false); return *chn != NULL;
154     case TX: *chn = iio_device_find_channel(get_ad9361_phy(ctx),
            get_ch_name("voltage", chid), true);  return *chn != NULL;
155     default: ASSERT(0); return false;
156     }
157 }
```

```
158
159 /* finds AD9361 local oscillator IIO configuration channels */
160 static bool get_lo_chan(struct iio_context *ctx, enum iodev d, struct
        iio_channel **chn)
161 {
162     switch (d) {
163      // LO chan is always output, i.e. true
164     case RX: *chn = iio_device_find_channel(get_ad9361_phy(ctx),
            get_ch_name("altvoltage", 0), true); return *chn != NULL;
165     case TX: *chn = iio_device_find_channel(get_ad9361_phy(ctx),
            get_ch_name("altvoltage", 1), true); return *chn != NULL;
166     default: ASSERT(0); return false;
167     }
168 }
169
170 /* applies streaming configuration through IIO */
171 bool cfg_ad9361_streaming_ch(struct iio_context *ctx, struct stream_cfg *cfg,
        enum iodev type, int chid)
172 {
173     struct iio_channel *chn = NULL;
174
175     // Configure phy and lo channels
176     printf("* Acquiring AD9361 phy channel %d\n", chid);
177     if (!get_phy_chan(ctx, type, chid, &chn)) { return false; }
178     wr_ch_str(chn, "rf_port_select",     cfg->rfport);
179     wr_ch_lli(chn, "rf_bandwidth",       cfg->bw_hz);
180     wr_ch_lli(chn, "sampling_frequency", cfg->fs_hz);
181
182     // Configure LO channel
183     printf("* Acquiring AD9361 %s lo channel\n", type == TX ? "TX" : "RX");
184     if (!get_lo_chan(ctx, type, &chn)) { return false; }
185     wr_ch_lli(chn, "frequency", cfg->lo_hz);
186     return true;
187 }
188 bool chkcfg_ad9361_streaming_ch(struct iio_context *ctx, struct stream_cfg
        *cfg, enum iodev type, int chid)
189 {
190     struct iio_channel *chn = NULL;
191
192     // Configure phy and lo channels
193     //  printf("* Acquiring AD9361 phy channel %d\n", chid);
194     if (!get_phy_chan(ctx, type, chid, &chn)) { return false; }
195 /*  rd_ch_lli( const struct iiochannel*, const char*, long long*)
196     chn A pointer to an iio_channel structure
197     attr A NULL-terminated string corresponding to the name of the attribute
198     val A pointer to a long long variable where the value should be stored*/
199     rd_ch_lli(chn, "rf_bandwidth",       &cfg->bw_hz);
200     rd_ch_lli(chn, "sampling_frequency", &cfg->fs_hz);
201     if (!get_lo_chan(ctx, type, &chn)) { return false; }
202     rd_ch_lli(chn, "frequency", &cfg->lo_hz);
203     return true;
204 }
205 //PLOT CONFIG
206 #define NUM_POINTS 5
207 #define NUM_COMMANDS 2
208
209 #define TABLE_SZ     1048576//1024*1024
210
```

```
211
212
213 /* simple configuration and streaming */
214 int main (int argc, char **argv)
215 {
216     //TABLE TO FIT THE SIGNAL
217     //const int tx_size = 1024*512;
218     int *tx_buffer;
219     //tx_buffer = malloc(f*(d ? 8 : 4));//[2*tx_size];
220
221     // Streaming devices
222     struct iio_device *tx;
223     struct iio_device *rx;
224
225     // RX and TX sample counters
226     size_t nrx = 0;
227     size_t ntx = 0;
228
229     // Stream configurations
230     struct stream_cfg rxcfg;
231     struct stream_cfg txcfg;
232
233     // Listen to ctrl+c and ASSERT
234     signal(SIGINT, handle_sig);
235
236     // RX stream config
237     rxcfg.bw_hz = MHZ(60.0);    // 2 MHz rf bandwidth
238     rxcfg.fs_hz = MHZ(60.0);    // 2.5 MS/s rx sample rate
239     rxcfg.lo_hz = GHZ(2.39); // 2 GHz rf frequency
240     rxcfg.rfport = "A_BALANCED"; // port A (select for rf freq.)
241
242     // TX stream config
243     txcfg.bw_hz = MHZ(60.0); // 1.5 MHz rf bandwidth
244     txcfg.fs_hz = MHZ(60.0);    // 2.5 MS/s tx sample rate
245     txcfg.lo_hz = GHZ(2.39); // 2 GHz rf frequency
246     txcfg.rfport = "A"; // port A (select for rf freq.)
247     int sigfreq= MHZ(2);
248
249
250
251     //int tx_size = txcfg.fs_hz/2;
252
253
254     tx_buffer = malloc(TABLE_SZ*sizeof(int));//[2*tx_size];
255
256     printf("* Creating the TX wave\n");
257     short ipart, qpart;
258
259     printf("* With size %i",TABLE_SZ);
260     for (int i = 0; i <TABLE_SZ; i++){
261     /*The latest version of the fmcomms IIO scope plug-in supports the TEXTU
            option valid range with the 'TEXTU' option is:
262         +--------------+-----------------+
263         |     Board    |      Range      |
264         +--------------+-----------------+
265         |fmcomms1      |  +/- 32767.0    |
266         |fmcomms2/3/4  |  +/- 2047.0     |
267         +--------------+-----------------+
```

```
268         */
269           ipart = (short)(amplitude *cos(2*M_PI *(double )i/
                  (double)(TABLE_SZ)));
270           qpart = (short)(-amplitude *sin(2*M_PI *(double )i/
                  (double)(TABLE_SZ)));
271
272           tx_buffer[i]= (int)ipart;
273           tx_buffer[i]= ((tx_buffer[i] << 16) | (((int)(qpart)) & 0xFFFF));
274
275       }
276       //exit(0);
277       printf("* Acquiring IIO context\n");
278       ASSERT((ctx = iio_create_default_context()) && "No context");
279       ASSERT(iio_context_get_devices_count(ctx) > 0 && "No devices");
280
281       printf("* Acquiring AD9361 streaming devices\n");
282       ASSERT(get_ad9361_stream_dev(ctx, TX, &tx) && "No tx dev found");
283       ASSERT(get_ad9361_stream_dev(ctx, RX, &rx) && "No rx dev found");
284
285       printf("* Configuring AD9361 for streaming\n");
286       ASSERT(cfg_ad9361_streaming_ch(ctx, &rxcfg, RX, 0) && "RX port 0 not
              found");
287       ASSERT(cfg_ad9361_streaming_ch(ctx, &txcfg, TX, 0) && "TX port 0 not
              found");
288
289       printf("* Checking AD9361 configuration\n");
290       struct stream_cfg chkcfg;
291       chkcfg.bw_hz = MHZ(0.0);
292       chkcfg.fs_hz = MHZ(0.0);
293       chkcfg.lo_hz = GHZ(0.0);
294       chkcfg.rfport = "A";
295       ASSERT(chkcfg_ad9361_streaming_ch(ctx, &chkcfg, TX, 0) && "TX port 0 not
              found");
296       printf("    - RF bandwidth %llu \n",chkcfg.bw_hz);
297       printf("    - Sample rate  %llu \n",chkcfg.fs_hz);
298       printf("    - LO frequency %llu \n",chkcfg.lo_hz);
299       printf("    - Tone freq = %d \n",sigfreq);
300       printf("    - Sin Table Size = %d \n",TABLE_SZ);
301       printf("    - RF port %s \n",chkcfg.rfport);
302
303
304       printf("* Initializing AD9361 IIO streaming channels\n");
305       ASSERT(get_ad9361_stream_ch(ctx, RX, rx, 0, &rx0_i) && "RX chan i not
              found");
306       ASSERT(get_ad9361_stream_ch(ctx, RX, rx, 1, &rx0_q) && "RX chan q not
              found");
307       ASSERT(get_ad9361_stream_ch(ctx, TX, tx, 0, &tx0_i) && "TX chan i not
              found");
308       ASSERT(get_ad9361_stream_ch(ctx, TX, tx, 1, &tx0_q) && "TX chan q not
              found");
309
310       printf("* Enabling IIO streaming channels\n");
311       iio_channel_enable(rx0_i);
312       iio_channel_enable(rx0_q);
313       iio_channel_enable(tx0_i);
314       iio_channel_enable(tx0_q);
315
316
```

```
317 #define BUFFER_SZ   1048576 //1024 *1024 size recomended on the template avoid
        to underfill the buffer.
318
319     printf("* Creating non-cyclic IIO buffers with 1 MiS\n");
320     rxbuf = iio_device_create_buffer(rx, BUFFER_SZ, false);
321     if (!rxbuf) {
322         perror("Could not create RX buffer");
323         shutdown();
324     }
325     txbuf = iio_device_create_buffer(tx, BUFFER_SZ, false);
326     if (!txbuf) {
327         perror("Could not create TX buffer");
328         shutdown();
329     }
330     int saltF= (int)((float)((float)txcfg.fs_hz/(float)TABLE_SZ)+0.5);
331     saltF=(int)(sigfreq/saltF);
332     saltF = saltF%(int)TABLE_SZ;
333     printf("* Starting IO streaming (press CTRL+C to cancel)  saltf=%d, fs=%f,
            TABLESZ=%d\n", saltF, txcfg.fs_hz, TABLE_SZ);
334     //exit(0);
335
336
337     if (saltF<=0){
338         saltF=1;
339     }
340     int i=0;
341     //int16_t zero=0;
342     printf("* With sin saltF = %d \n",(int)(saltF));
343     printf("* With sin f = %d Hz\n",(int)((chkcfg.fs_hz/(TABLE_SZ/saltF))));
344
345     /* All the file setings are set to store a binary file with the RX data to
            check the signal*/
346     FILE *write_ptr;
347     write_ptr= fopen("rx.bin","wb");//w-> write b-> binary
348
349     while (!stop)
350     {
351         ssize_t nbytes_rx, nbytes_tx;
352         char *p_dat, *p_end;
353         int *p_dat_int;
354         ptrdiff_t p_inc;
355
356         // Schedule TX buffer
357         nbytes_tx = iio_buffer_push(txbuf);
358         printf("nbytes_tx=%d\n", nbytes_tx);
359         if (nbytes_tx < 0) { printf("Error pushing buf %d\n", (int)
                nbytes_tx); shutdown(); }
360
361         // Refill RX buffer
362         nbytes_rx = iio_buffer_refill(rxbuf);
363         if (nbytes_rx < 0) { printf("Error refilling buf %d\n",(int)
                nbytes_rx); shutdown(); }
364
365         // READ: Get pointers to RX buf and read IQ from RX buf port 0
366         p_inc = iio_buffer_step(rxbuf);
367         p_end = iio_buffer_end(rxbuf);
368         p_dat = (char *)iio_buffer_first(rxbuf, rx0_i);
369         //int nmemb = ((p_end-p_dat)/p_inc);
```

```
370         for (p_dat = (char *)iio_buffer_first(rxbuf, rx0_i); p_dat < p_end;
                p_dat += p_inc) {
371             const int16_t i = ((int16_t*)p_dat)[0]; // Real (I)
372             const int16_t q = ((int16_t*)p_dat)[1]; // Imag (Q)
373             int nbytes_rx_wr_file = fwrite((const void*)&i,sizeof(const
                    int16_t),1,write_ptr);
374             nbytes_rx_wr_file = fwrite((const void*)&q,sizeof(const
                    int16_t),1,write_ptr);
375             if (nbytes_rx_wr_file < 0) { printf("Error writting RX samples to
                    file\n"); shutdown(); }
376         }
377
378
379
380         // WRITE: Get pointers to TX buf and write IQ to TX buf port 0
381         p_inc = iio_buffer_step(txbuf);
382         p_end = iio_buffer_end(txbuf);
383         p_dat = (char *)iio_buffer_first(txbuf, tx0_i);
384
385         for (p_dat; p_dat < p_end; p_dat += p_inc) {
386             printf("pdat=%p, pend=%p, pinc=%d\n", p_dat, p_end, p_inc);
387             // 12-bit sample needs to be MSB alligned so shift by 4
388             //
                    https://wiki.analog.com/resources/eval/user-guides/ad-fmcomms2-ebz/software/bas
389             /* In binary format each I or Q word is in 16-bit signed format.
                    The AD9361 bus-width is 12-bit. Therefore Bit# 11 becomes Bit#
                    15. So they are shifted by 4 and the lower 4 bits are ignored.
390             Buffer Bit  15  14  13  12  11  10  9   8   7   6   5   4   3   2
                      1   0
391             AD9361 Bit  11  10  9   8   7   6   5   4   3   2   1   0   X   X
                      X   X
392 */          //PSEUDORANDOM NOISE GENERATOR
393             ((int16_t*)p_dat)[0] = ((int16_t)(rand() %2047)); // Real (I)
394             ((int16_t*)p_dat)[1] = (int16_t)(0); // Imag (Q)
395             //TRANSMITTING THE SINE
396
397 /*          ((int16_t*)p_dat)[0] = ((int16_t)((tx_buffer[i] >> 16)&0xFFFF));
        // Real (I)
398             ((int16_t*)p_dat)[1] = (int16_t)((tx_buffer[i])&0xFFFF);     //
                    Imag (Q)
399 */
400             i=i+saltF;
401             if(i >= TABLE_SZ){
402                 i=0;
403                 //zero ++;
404                 //zero = zero%2;
405             }
406
407
408         }
409
410         // Sample counter increment and status output
411         nrx += nbytes_rx / iio_device_get_sample_size(rx);
412         ntx += nbytes_tx / iio_device_get_sample_size(tx);
413         printf("\tRX %8.2f MSmp, TX %8.2f MSmp\n", nrx/1e6, ntx/1e6);
414     }
415
416     shutdown();
```

```
417
418     return 0;
419 }
```

./appendix/ad9361–iiostream_txrxok.c

APPENDIX B. SIGNAL CHECKER CODE IN PYTHON

```
1 import binascii
2 import struct
3 import numpy as np
4 import matplotlib.pyplot as plt
5 import scipy.fftpack
6
7 with open("rx.bin", "rb") as binary_file:
8     print("LOADING SIGNAL")
9     data = binary_file.read(4)
10    signal =[]
11    while len(data)>= 4:
12        #print(binascii.hexlify(data))
13        f = struct.unpack('hh',data)
14        #print(complex(f[0],f[1]))
15        signal.append(complex(f[0],f[1]))
16        data = binary_file.read(4)
17    print("PREPARING FFT")
18    # Number of samplepoints
19    N =  8192
20    # sample spacing
21    T = 0.000000008 # 60 MHz
22    #T = 0.00000025 # 2MHz
23    x = np.linspace(0.0, N*T, N)
24    y = signal[:N]#np.sin(50.0 * 2.0*np.pi*x) + 0.5*np.sin(80.0 * 2.0*np.pi*x)
25    yf = scipy.fftpack.fft(y)
26    xf = np.linspace(0.0, 1.0/(2.0*T), N/2)
27    print("PLOTING FFT")
28    fig, ax = plt.subplots()
29    ax.plot(xf, 2.0/N * np.abs(yf[:N//2]))
30    plt.show()
```

./appendix/checkRX.py